IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Temporal Graph Traversals using Reinforcement Learning with Proximal Policy Optimization

SAMUEL HENRIQUE SILVA[1,2], ADEL ALAEDDINI[3], PEYMAN NAJAFIRAD [1,2,4]

[1]Secure AI & Autonomy Laboratory, University of Texas at San Antonio, San Antonio, TX 78249, USA
[2]Department of Electrical and Computer Engineering, University of Texas at San Antonio, San Antonio, TX 78249, USA
[4]Department of Information Systems and Cyber Security, University of Texas at San Antonio, San Antonio, TX 78249, USA
[3]Department of Mechanical Engineering, University of Texas at San Antonio, San Antonio, Texas, TX 78249, USA

Corresponding author: Peyman Najafirad (e-mail: peyman.najafirad@utsa.edu).

The authors gratefully acknowledge the use of the services of Jetstream cloud, funded by National Science Foundation, United States award 1445604.

ABSTRACT Graphs in real-world applications are dynamic both in terms of structures and inputs. Information discovery in such networks, which present dense and deeply connected patterns locally and sparsity globally can be time consuming and computationally costly. In this paper we address the shortest path query in spatio-temporal graphs which is a fundamental graph problem with numerous applications. In spatio-temporal graphs, shortest path query classical algorithms are insufficient or even flawed because information consistency can not be guaranteed between two timestamps and path recalculation is computationally costly. In this work, we address the complexity and dynamicity of the shortest path query in spatio-temporal graphs with a simple, yet effective model based on Reinforcement Learning with Proximal Policy Optimization. Our solution simplifies the problem by decomposing the spatio-temporal graph in two components: a static and a dynamic sub-graph. The static graph, known and immutable, is efficiently solved with A^* algorithm. The sub-graphs interconnecting the static graph have unknown dynamics and we address such issue by estimating the unknown dynamic portion of the graph as a Markov Chain which correlates the observations of the agents in the environment and the path to be followed. We then derive an action policy through Proximal Policy Optimization to select the local optimal actions in the Markov Process that will lead to the shortest path, given the estimated system dynamics. We evaluate the system in a simulation environment constructed in Unity3D. In partially structured and unknown environments, with variable environment parameters we've obtained an efficiency 75% greater than the comparable deterministic solution.

INDEX TERMS Machine Learning, Graphs, Markov-Chain, Deep Reinforcement Learning, Path-Planning

I. INTRODUCTION

WITH the rapid increase of data availability, and the increased use of data analytics for decision making, the necessity of efficient techniques for data querying rise. Different from standard data structures, Graphs (a.k.a. networks) have been used to organize information in various areas including biology [1], social sciences [2], linguistics [3], and even path planning in robotics (grid networks) [4]. Moreover, the search of information in such networks is more computationally expensive as the correlation between the nodes increase [5]. In addition such networks, even though sparse in a global sense, tend to be dense in a local sense (nearby nodes) [6]. Even though more paths and links exist, there is more variability as we consider the dynamicity of the information, and consequently, the constant change of the network structure [7]. Such dynamic graphs are theoretically attractive, as well as computationally challenging. A Graph is defined as dynamic when the data structure components (such as vertices, edges, weights) change along with the evolution of time. In general, weights are the most time-dependent entities, but in many applications, the nodes of the graph can completely change due to the dynamic nature of the data.

In graph databases, a very important problem is the *st-connectivity* problem. Given a graph $\mathcal{G}$, an input node s and a target node t, the *st-connectivity* problem is to define if

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2985295, IEEE Access

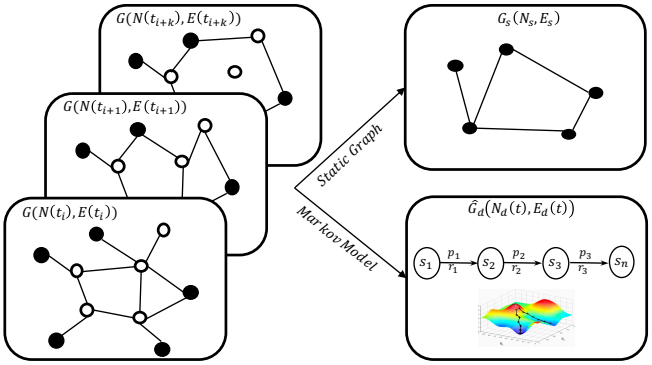


FIGURE 1: The agent learns the spatial-temporal graph dynamics, and the relation between the dynamic nodes are abstracted into probability distributions mapping embedded states to optimal actions in the environment.

there's a path in $\mathcal{G}$ that connects s and t [8]. Such problems translate directly to real-world applications such as robotic grid networks, social network advertisement suggestions, and real-time information queries from stock markets. Planning for such systems involves dealing with problems not faced in simpler domains. A problem of the information query in such domains is the increased number of ramifications of vertices and nodes from a graph representing such systems. As an example, the graph representing both the dynamics of a robot and the dynamics of the environment grows exponentially as one models all the position and velocity variations that a robot is allowed in each position of the environment. Moreover, requesting a short path in such an environment would require each time step re-planning [9]. Such requirements make the short path query prohibitive in such large structures, even more so if we consider that accurate models of such systems are hard to obtain and quickly become outdated due to the dynamic nature of the system.

The task of guiding the robot with a network graph becomes more challenging as we consider that the environment is imprecise, vast, dynamic, and partially non-structured. To capture such restrictions, the agent has to rely on real-time sensory information captured from the environment, which in general contains noise and uncertainty. As a consequence, the control scheme of the agent under such rough conditions should be adaptive. To achieve such level of adaptiveness, an agent with a path generated upon a graph network would require both a very detailed discretization of the space and constant route re-planning and searching to account for unexpected environmental changes. These and other conditions change the representation of the environment, requiring it to be represented not as a grid, but a dynamic navigation mesh in which the adjacency graphs usually have highly irregular edge costs connecting a much larger number of nodes. Such configuration increases the complexity of the system, making it impossible to compute safe and efficient paths in real-time. Alternatively, local path planners rely on local sensing and reaction to avoid collisions in possibly unknown environments, which means it applies to short length paths in which no high-level decision (like goal localization) should be taken. Typical examples of local path planners include the potential field-based navigation [10], dynamic window approaches [11], and sampling-based approaches [12]. However, as we increase the distance between waypoints, add more adverse scenarios, and increase constraints, local path planners based on strict deterministic conditions tend to fail, due to their susceptibility to local minima. Probabilistic approaches can avoid such problems due to taking actions based on a probability distribution rather than a deterministic function. Learning-based methods have been shown to present better and more compliant results for navigation in imprecise, dynamic, and partially nonstructured environments [13], as they succeed in incorporating the uncertainty and dynamicity of such environments and apply adequate actions for safe behavior.

On the other hand, learning-based methods such as Reinforcement Learning (RL) can be tough to converge in large and complex scenarios, which would make it hard to train an end-to-end agent capable of navigating in a given environment. As a consequence, in this work, we combine the stochastic nature of decision policies learned with the RL approach to simplify the discretization requirements in the environment. More specifically, we extract portions of a graph network and let the navigation in these points to be handled by a non-deterministic policy.

In this paper we propose the use of graph decomposition [14], to reduce the complexity of the short path query in a graph network, based on the importance, and the static characteristic of certain nodes [15], [16]. Our proposed model is simple, yet effective in addressing the complexity of the short-path query, based on Reinforcement Learning with Proximal Policy Optimization. Unlike D^* [17] or A^* [18] which focus only on immutable graphs, or Rapidly Exploring Random Trees (RRT) [19], Dynamic Domain RRT (DDRRT) [20], and Probabilistic Road Maps (PRM) [21] which requires an every-step update of the knowledge, our solution simplifies the problem by decomposing the spatio-temporal graph into two components - a static and a dynamic sub-graph - and solves them simultaneously. The static graph, known and immutable, is efficiently solved with A^* algorithm. The sub-graphs interconnecting the static graph have unknown dynamics and we address such issue by estimating the unknown dynamic portion of the graph as a Markov Chain. We then, derive an action policy through Proximal Policy Optimization to select the local optimal actions in the Markov Process that will lead to the shortest path, given the estimated system dynamics. With this approach and a known policy, there is no need for every-step re-planning. The policy optimization was performed in a cloud environment to reduce the time required [22], [23].

The main contribution of this paper consists of providing an efficient solution to the short-path query problem in spatio-temporal graphs using graph-decomposition and state action correlation estimation through RL framework, with the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2985295, IEEE Access

following key aspects:

1) we address the problem of short-path query in spatio-temporal graphs without the need of every-step replanning, which significantly reduces the computation load;
2) we propose a method for estimating the dynamic changes in the state action correlation of the spatio-temporal graph through a reinforcement learning framework, and optimize the state to action policy using Proximal Policy Optimization;;
3) we show the efficiency of the approach by presenting a thorough analysis of corner case scenarios by implementing the decision making algorithm in multiple real-time 3D simulated environments, created in Unity3D.

II. RELATED WORK AND BACKGROUND

A single socially aware agent path planning with collision avoidance and global orientation can be formulated as a sequential decision-making problem in a deep reinforcement learning framework with dynamic goals and observations. We use A^* algorithm to generate goals, and we train our agent, using Proximal Policy Optimization (PPO) [24] to follow those way-points while avoiding static, dynamic, and human obstacles. We use a joint state representation to capture crowd behavior and enhance the social skills of the robot.

A. RELATED WORK

As mentioned, Path Planning is a widely explored area, but with still many aspects to improve. Several algorithms have already been developed for UAV's [25]–[27], underwater vehicles [28]–[30], and ground vehicles [31]–[33]. For controlled environments and restricted conditions, several algorithms have been synthesized for path planning in [34]. Aside from the standard path planning algorithms proposed in [35], new approaches and work have been published recently showing that it's a hot and demanding field.

In recent years, the publications on Path Planning either focus on a local path planner or a global path planner separately. In the realm of global planners, mapless navigation has drawn some attention. In [36] the global path planner focuses on the recognition of pre-trained landmarks for the global localization; even though there is no need for a map, the system requires a dataset with global coordinates of the landmarks. Differently in [37], the authors modify $A*$ path planning to make the search more efficient in large graph structures. In [38], a genetic algorithm is provided to find a global path in a grid environment. In [39], a CNN method is used on planetary images to derive a global path for interplanetary rovers. Moreover, when dealing with non-adversarial multiple agents, global planners focus on the consensus of such agents. In [40], [41], the author builds a global controller with the objective of cooperation among the agents which compose the systems, in a latter publication [42], instead of fully modelling the dynamics of the system, the uncertainties were dealt with an adaptative neural network.

New and innovative results in local path-planning were proposed in [43]–[47], but none of these focused on navigation in crowded environments. A new work in local and a global path planners [48], the authors integrate a Generative Adversarial Network as a decision making planner to evaluate the condition of the current path and decide if it is safe to proceed or not. In [49], the authors have integrated a local navigator based on a CNN called intention-net with a global navigator to generate an indoor navigation system. Another method was presented in [50],the PRM-RL, which consists of a hierarchical method for long-range navigation tasks that combines sampling-based path planning with reinforcement learning.Different from their method, we do not vary the number of way-points known in the environment, we leverage the pre-existing knowledge of the intersections, and more important, while in PRM-RL the authors restrict their obstacles to be only static, in our work we train and evaluate our system under hard dynamic conditions.

In [51], a RL approach was proposed to solve a mapless navigation problem in the crowd, but different from our work, the authors have only focused on the local navigation without considering the global planner. In [52], a deep inverse reinforcement learning approach was proposed to learn pedestrian behavior from a dataset, but no path planner was proposed. In [13], [53], the authors in a previous work encoded the states of a fixed amount of surrounding agents with the states of the agent to capture the behavior of other agents. In the latter work, an LSTM was proposed to encode the state of surrounding agents, and consequently consider as many surrounding agents as robot detects. However, these approaches are restricted to a local planning, whereas our work integrates a deterministic decision-making global path planner that takes into consideration a priory knowledge of the environment and enables navigation in more complex and broader environments with a local non-deterministic approach, which integrates the state of surrounding people and predicts, before action, the behavior of surrounding obstacles.

B. REINFORCEMENT LEARNING FRAMEWORK

A Markov Decision Process (MDP) is a mathematical description of sequential decision making [54]. An agent observes the environment in the form of states. In each state, the agent takes an action which evolves the environment into another state, receiving or not a reward in the process. In MDP's, only the current state and action influence the probability of the next state, action, and reward. Formally, MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, r, \gamma \rangle$, where $\mathcal{S} = \{s_1, s_2, , ..., s_k\}$ is the state space, $\mathcal{A} = \{a_1, a_2, ..., a_n\}$ is the action space, $\mathcal{T}$ is the set of transition probabilities, such that $T(s, a, s') = p(s'|s, a)$ is the transition probability of going to state s' by taking action a on state s, $\gamma \in [0, 1)$ is the discount factor and $r : \mathcal{S} \times A \rightarrow \mathbb{R}$ is the reward function where $r(s, a)$ is the reward obtained by taking a in s. A stochastic policy $\pi_\theta : \mathcal{S} \rightarrow \mathcal{A}$ specifies the action distribution conditioned to the state s and parameterized by the parameters θ.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2985295, IEEE Access

MDP's are the bases for RL framework. The RL framework makes it possible to underlay the unknown state probability distribution and optimize the policy towards goal accomplishment. Several methods have been derived to solve the RL problem. As stated in the previous section, we will focus on gradient policy methods as they have become prevalent recently and have shown significant improvements towards those based on Value or Q-value functions. In RL, policy gradient methods, are learning algorithms that focus on learning a parameterized policy which describes the probability distribution of actions over the state space. We define the policy as

$$\pi(a|s,\vec{\theta}) = P\{A_t = a|S_t = s, \vec{\theta} = \theta\}.$$

The focus of policy gradient methods is to build an estimator of the policy gradient and using a stochastic gradient ascend, adjust the weights of the policy towards the maximum rewards. An unbiased estimator for the gradient is given by

$$\hat{g} = \hat{\mathbb{E}}_t[\bigtriangledown_\theta log \pi_\theta(a_t|s_t)\hat{A}_t],$$

where $\hat{A}_t$ is the estimator of the advantage function at time step t. In this case, $\hat{\mathbb{E}}$ is the expected value over a finite trajectory of the system. We define a trajectory $\tau = \{(s_1, a_1, r_1), (s_2, a_2, r_2), ...\}$ of the system as a sequence of state, action, and rewards. In order to define the advantage, $\hat{A}_t$, function we define:

$$\eta(\pi) = \mathbb{E}_{s_0,a_0,...}[\sum_{t=0}^{\infty} \gamma^t r(s_t)] \tag{1}$$

$$Q_\pi(s_t, a_t) = \mathbb{E}_{s_{t+1},a_{t+1},...}[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l})] \tag{2}$$

$$V_\pi(s_t) = \mathbb{E}_{a_t,s_{t+1},...}[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l})], \tag{3}$$

where $\eta(\pi)$ is the expected discounted reward, $Q_\pi(s_t, a_t)$ is the state-action value function and $V_\pi(s_t)$ is the value function. With Equation 2 and Equation 3, we define the advantage function as:

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s),$$

where $a_t \sim \pi(a_t|s_t), s_{t+1} \sim P(s_{t+1}|s_t, a_t)$ for $t \geq 0$.

Following with the gradient optimization for continuous state space, in [55] the authors presented a framework, named Deep Deterministic Policy Gradient (DDPG), to address the application of Deep Reinforcement Learning in continuous control of objects. However, the main issue of DDPG is choosing the right step size for the optimization of the policy. A small step size would make the convergence too time-consuming, and one too large would lead to the optimization being overwhelmed by noise, reducing the performance of the policy drastically.

When Trust Region Policy Optimization was proposed in [56], it aimed to propose a method to update the parameters in a manner that would guarantee policy improvement; namely, we want Equation 1 to be an always increasing function. In order to guarantee policy improvement, the old policy, π_{old}, is updated in a trusted region.

More precisely, Equation 1 is changed, based on [57], so that the expected discount reward of a sampled sequence is just an increment to the old policy, as shown by

$$\eta(\tilde{\pi}) = \eta(\pi_{old}) + \mathbb{E}_{\tau\sim\tilde{\pi}}[\sum_{t=0}^{\infty} \gamma^t A_{\pi_{old}}(s_t, a_t)],$$

which can be locally approximated around a region within step size δ,

$$L(\tilde{\pi}) = \eta(\pi_{old}) + \sum_s \rho_\pi(s) \sum_a \tilde{\pi}(a|s) A_{\pi_{old}}(s, a)],$$

in terms of optimizing the parameterized policy, TRPO does so by maximizing a surrogate objective function constrained to the max step size (the trust region). Specifically,

$$\begin{aligned} &\underset{\theta}{\text{maximize}}\ \hat{\mathbb{E}}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t] \\ &\text{subject to}\ \hat{\mathbb{E}}_t[KL[\pi_{\theta_{old}}(.|s_t), \pi_\theta(.|s_t)]] \leq \delta, \end{aligned} \tag{4}$$

where KL is the Kullback-Leibler divergence [58] function for distributions.

In TRPO, Equation 4 can also be seen with the constraint added to the objective function as penalty weighted by a constant β, as seen in:

$$\max_\theta \hat{\mathbb{E}}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t] - \beta KL[\pi_{\theta_{old}}(.|s_t), \pi_\theta(.|s_t)]. \tag{5}$$

Eq. 5 is the starting point for Proximal Policy Optimization method, which proposes two methods for the policy updates. The first method proposes an adaptive weight for the KL divergence penalty. The objective is to achieve a target value of the divergence, d_{targ} each policy update. The following rules adapt the value of β:

1) Compute $d = \mathbb{E}_t[KL[\pi_{\theta_{old}}(.|s_t), \pi_\theta(.|s_t)]]$
2) $\beta = \begin{cases} \beta/2 & \text{if, } d_{targ} \leq d_{targ}/1.5 \\ \beta \times 2 & \text{if, } d_{targ} \geq d_{targ} \times 1.5 \end{cases}$

A slightly different approach for the penalty weight brought better overall results than previous techniques. The proposed approach was a clipped version of Equation 5. In this clipped version, the objective function is defined as

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta))\hat{A}_t, \text{clip}(r_t(\theta)), 1-\epsilon, 1+\epsilon)\hat{A}_t] \tag{6}$$

where ϵ is a tunable hyperparameter. Under this condition, the final objective is a lower bound (pessimistic) on the unclipped objective, in which the probability ration is included when it would make the objective worse and not considered when it would improve the objective function.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2985295, IEEE Access

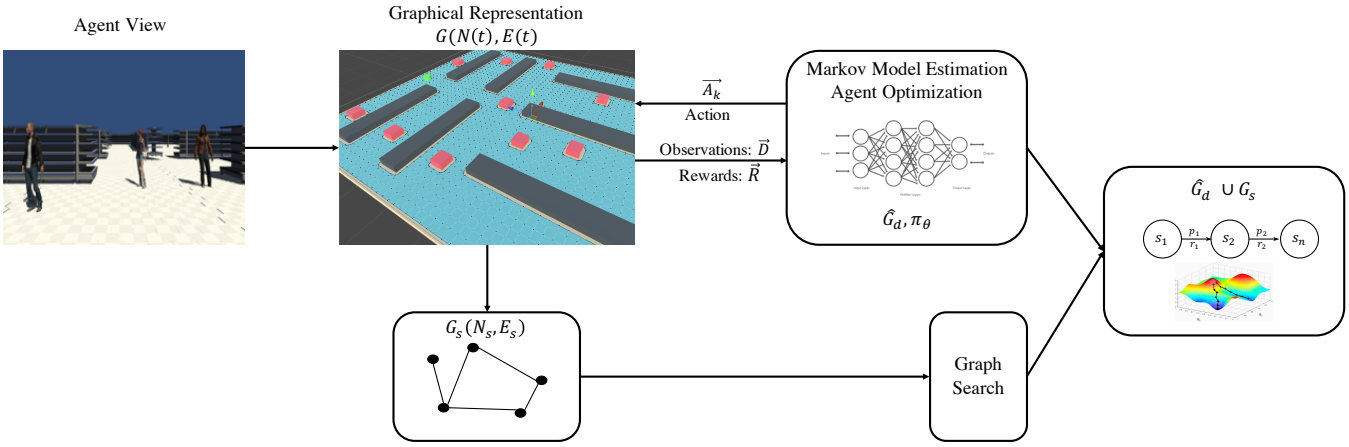


FIGURE 2: The agent view and the flow of graphical representation and estimation. Given the Spatio-Temporal representation of the system as $G(N(t), E(t))$ and $G_s(N_s, E_s)$, the transition probability model is estimated using RL framework. The static graph G_s is searched with deterministic algorithm. With the estimated model, the optimized policy π_θ, and the set of waypoints, the desired path is generated.

III. PROBLEM FORMULATION

In this section, we formally define our problem linking the *st-connectivity* to the robotics path planning, and define the graph decomposition proposed to solve such a problem. The *st-connectivity* problem, consists in determining if there is a connection between nodes s and t, and, under the assumption that such a connection exists, determines in real-time a shortest path query. We solve the short path query in dynamic graphs with graph decomposition, an estimation of the dynamics of the agent and environment with the RL framework.

We model the mapping of the environment, the environment dynamics, and agent dynamics as a graph $\mathcal{G}$. The nodes of $\mathcal{G}$ represent entities in the system (such as position in a grid, people in social networks, and the information in databases) and the links between these entities models the connection that exists between this information and the cost of a path between them. As we assume these entities are dynamic, the weights defining the cost of traversing from node n_i to n_{i+1}, become a function of time, $w(t)$. Furthermore, $\mathcal{G}$ incorporates the dynamics of the agent, as mentioned in [59], where $\mathcal{G}$ is extended to incorporate the state of the agent (a.k.a. state lattices) and its node transition limitations. With the dynamics, time, and lattices assumption, we can define the model of the system as,

$$\mathcal{G} = G(N(t), E(t)), \tag{7}$$

in which, $N(t)$ and $E(t)$ are time-dependent functions that incorporate the dynamicity of the whole system. Such a model restricts conventional solutions for graph traversing or would require trial and error path searches at execution time, which would reduce the performance of the system drastically.

To solve such a problem, assume $\mathcal{G}$, is composed of two or more edge disjoint sub-graphs, $\mathcal{G} = \{G_1, G_2, \dots\}$ in which, we can decompose the system in, at least two structurally different instances of the system: *(i)* $G_1(N_s, E_s)$ - the static and immutable graph of the system, and *(ii)* $G_2(N_d(t), E_d(t))$ - the dynamic graph of the system. $G_1(N_s, E_s)$ is static and assumed to be known by the user in the form of maps(path planning), interpersonal relations(social networks), or information correlation (databases).

Under such conditions, $G_1(N, E)$ the *st-connectivity* problem can naturally be solved with standard graph search techniques. Still, $G_2(N_d(t), E(t))$ remains unsolved. Given the random nature of $G_2(N_d(t), E(t))$, to determine with precision the value of the weights of the edges and the existence of nodes, requires approximation, which can be estimated if we assume, that:

$$G_2(N_d(t), E(t)) \propto S^a(t) \,\&\, S^e(t),$$

where $S^a(t) = \{s_1^a, s_2^a, \dots\}$ and $S^e(t) = \{s_1^e, s_2^e, \dots\}$ are respectively the states of the agent and the environment. Here, we use the symbol $\propto$ to indicate a relation between the dynamic graph and its decomposition, even though the exact transformation and relation is unknown. It is reasonable to assume that estimating the agent's model and environment model to a certain level of confidence, will suffice to navigate in the system. Under that perspective we define

$$\hat{\Pi} : (S^a, S^e) \in R^n \mapsto \vec{V}_\theta \in R^m,$$

in which, n is the order of the dynamics of the combined agent and environment system, and m is the number of parameters used to parameterize the system function. Such mapping is highly dependent on the system. As we try to capture the dynamics of a highly probabilistic system, it is logical to model the system as a Markov chain, in which we compose the state of the system, with the fusion of the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2985295, IEEE Access

dynamics of the environment and the agent. The combined state S^c is defined as $S^c = \{\{s_1^e, s_1^a\}, \{s_2^e, s_2^a\}, ...\}$.

The dynamics of the system is modeled by estimating from experience and the transition probabilities between the states of the system. Experience is defined as the trajectory, or sequences of state/action/reward, $\tau = (s_1, a_1, r_1), (s_2, a_2, r_2)$. Moreover, we use a reinforcement learning approach to capture the dynamics of the interaction between the environment and the agent, and consequently map the states to the actions through the parameterized policy π_θ. The collection of trajectories of the agent in the environment is used to generate the gradient estimators, which leads to the optimization of the policy parameters θ, which we define learning by experience. More specifically, we estimate the chain of transition probabilities $s^c \rightarrow s^{c\prime}$ from the system. Assuming a uniform distribution over the initial states, $p_0 = \mathcal{U}\{s^c = s_0^c\}$, and assuming that the state incorporates enough information of the system, we can claim the Markov property, in which the future state depends only on the current state. As a consequence, we can define the probability of the system being in a certain state in time T as:

$$P(S_T^c) = p_0 \prod_{t=0}^{T-1} P(S_{t+1}^c | S_t^c) \tag{8}$$

Moreover, with the system modeled as a chain of transition probabilities independent of past states, the *st-connectivity* problem resumes to defining a set of actions, which maximize the probability of reaching state s_n^c, which connects to the next node of $G_1(N_s, E_s)$. For that, as described in subsection II-B, we define a parameterized policy $\pi_\theta : \mathcal{S} \rightarrow \mathcal{A}$ which maps the action distribution over the states. Moreover, with the knowledge of the objectives of the system, a reward function is also defined as $r(s, a)$. The key aspect to estimate the desired behavior of the system, specifically the shortest path query is to define the objectives of the agent. Such objectives are application dependent and need to be translated to the mathematical formulation of the implementation. In this work, based on the formulation from [60], we've defined the rewards function $r_{total}(s, a, s')$ as a composition of K reward functions that supports or punishes specific desired or undesired behaviors of the agent.

$$r(s, a, s') = \sum_{k=1}^{K} r_k(s, a, s')$$

with these terms derived from the graph $\mathcal{G}$, we can directly utilize the described RL framework, described from Eq. 1-6 to estimate the states of the system, and derive π_θ.

We now, apply such formulation in a robotic path planning scenario with complex and adversarial environments, involving static, dynamic, and crowded scenarios in which we divide the Graphs into two control schemes: *(i)* - The immutable graph G_1, represents the general knowledge of the environment and is used by a global controller for the global navigation, and *(ii)* - the dynamic graph, G_2, is abstracted by a RL framework, in a local controller scheme, in which we train our policy using PPO and define a set of joint states to learn social behavior and improve social compliance.

Algorithm 1 Spatio-Temporal Graph Abstraction

Input:
- $G = (N(t), E(t))$
- $G_s = (N_s, E_s)$

for Number Episodes $\leq$ MAX Episodes **do**
  while Each $E_s \in G_s$ **do**
    Run $\pi_{\theta_{old}}$ for T time-steps
    Compute advantage estimates $\hat{A}_1, ..., \hat{A}_T$
  Optimize L from Eq. 6 wrt θ
  $\theta_{old} \leftarrow \theta$
  $\hat{G}_d^{old} = \hat{G}_d$
return

For illustration, in the problem definition, we assume an environment in which the agent should navigate through paths (corridors, streets) which intersect in known locations. We represent the whole, static and known environment as an undirected graph $G(N, E)$, in which the intersections of the corridors are the vertices N, and the corridors are the edges E. Each of the vertices represents a fixed Cartesian pair $N_i = (x_i, y_i)$. Each Cartesian pair (x_i, y_i) is the exact location of vertice N_i in the map of the environment. A sequence of vertices $\mathcal{N} = \{N_1, N_2, ..., N_k)\}$ defines the desired path along the known environment.

Different from [61], our system does not propose that the global planner should generate very close waypoints and the local controller would guide the agent through minimal distances. We propose a global planner that will guide the agent in crucial portions of the environment, like intersections, directing it to one of the available directions according to the designated path. Meanwhile, the local planner avoids collisions, maintains system safety, and drives the agent towards N_{i+1}.

This problem can easily be visualized in a warehouse or supermarket scenario, in which the agents (robots) need to reach specific shelves to collect or store specific products. While such a problem can be solved easily for a static scenario, where no interaction with humans happens and a central controller is available, when such conditions are not met, robots are forced to go through an every-step re-planning, which is computationally costly and reduces the time available to respond to immediate danger. Under such conditions, with the proposed framework, a way to model the system, is assuming the static graph G_s to model the immutable interest points (shelf locations, product location, turns). On the other hand, $\hat{G}_d$, the estimated Markov model, establishes a relation between the actions of the agent and the observations it can collect from the environment.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2985295, IEEE Access

A. GLOBAL PATH PLANNING

For the global navigation problem, we consider the path mapping, where the robot needs to navigate from origin O to goal, G through a known environment. We assume a search in an indoor environment, at this level with only the environment original obstacles, namely walls, shelves, and other items that were part of the initial environment. First, we define in the scene points of interest or waypoints through which the goal can be achieved, but not considering the specifics of local navigation like velocity, collision avoidance, and obstacle avoidance. The disposition of intersections and corridors created by the ambient structures are taken into consideration to build a graph that describes the existence, or lack thereof, of a path between places in the environment.

As proposed in [62], the adjacent way-points form a graph structure. In order to have a more goal-oriented movement, we implement the path search algorithm A^*. In this specific implementation, we use the Manhattan Distance heuristic in order to capture the Cartesian aspect of our environment. At each iteration of its main loop, A^* needs to determine which of its partial paths to expand into one or more longer paths. It does so based on an estimate of the cost (total weight) to get to the goal node. Specifically, A^* selects the path that minimizes:

$$f(n) = g(n) + h(n),$$

where n is the last node on the path, $g(n)$ is the cost of the path from the start node to n, and $h(n)$ is a heuristic that estimates the cost of the cheapest path from n to the goal. The heuristic is problem-specific. As we are dealing with a grid, which allows four directions of movement, the most suitable heuristics uses Manhattan distance (L_1). The cost of moving from point a to b is defined as the distance between the vectors $\vec{p} = (p_1, p_2, ..., p_k)$ and $\vec{q} = (q_1, q_2, ..., q_k)$, which is:

$$h_1(\vec{p}, \vec{q}) = \beta(\sum_{i=1}^{k} |p_i - q_i|)\beta,$$

in a $x - y$ plane the cost can be reduced to $h_1(\vec{p}, \vec{q}) = \beta(|p_x - q_x| + |p_y - q_y|)$. In both cases, β represents the minimum cost to translate from the current node to the adjacent one.

The graph search generates a tree, which defines a path plan to be followed. Even if the generated tree may lead towards the goal, it does not take into consideration the set of decisions that should be taken between waypoints. The navigation between waypoints is the core of our work and is described in details in subsection III-B.

B. LOCAL CONTROLLER

With the A^* search, we generate a set of waypoints that should be followed, but so far we have not discussed the navigation in-between waypoints. Before discussing the control method, we define the second-order model, S of our agent as a double-integrator, as proposed in [63],

$$S(t) = \begin{bmatrix} \dot{r_x} \\ \dot{r_y} \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \end{bmatrix}, \quad \begin{bmatrix} \dot{v_x} \\ \dot{v_y} \end{bmatrix} = \begin{bmatrix} u_x \\ u_y \end{bmatrix}. \tag{9}$$

In Equation 9, the system is independently controlled on x and y directions. The controller $u_q(t)$ applies a force to the agent $S(t)$ in order to direct towards the reference point.

In Figure 3a, the feedback branch with the observations is not subtracted of the input vector X_k, and instead both are fed to the controller that decides its current state and define the best action pair to be applied, $u_k = [u_x, u_y]^T$.

The local controller is formulated as a sequential decision making entity trained in a reinforcement learning framework [54], [64], [65]. In order to drive the agent from position X_{k-1} to X_k, a decision-making algorithm was trained in a straight line environment initially with static obstacles, and later with the crowd. With such a scenario, the agent would initially learn about the goal achievement, and in a second moment would enhance crowd avoidance by incorporating behavior learning.

As shown in Figure 3b, the RL framework does not react to the difference of current agent position and the goal only. On the contrary, this information is used to compose the state $S_i \in \mathcal{S}$ of the agent. We define the state as

$$S = \{(x_a - x_g), (y_a - y_g), \dot{x}_a, \dot{y}_a\}, \tag{10}$$

in which $(r_a - r_g)$ represents the vector distance between current position r_a and the goal position r_g. In the crowd environment, the state of the agent for the RL framework is not enough to generate the desired actions, as we consider the environment to be continuously changing and evolving. As for the system to incorporate the crowd behavior, the system state incorporates position and velocity of the nearest human

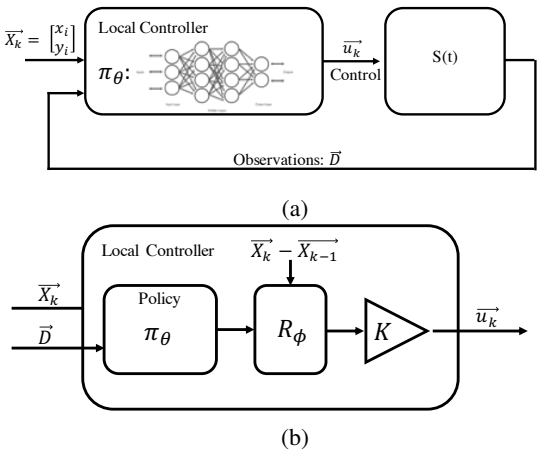


FIGURE 3: Local controller system. (a) - The control input u_k is applied to the system $S(t)$, which reacts, generating the observation vector $\vec{D}$. Different from regular feedback, the observation vector is fed as whole to the controller. (b) - the local controller performs a rotation in the output of π_θ based on the difference from $\vec{X}_k$ and $\vec{X}_{k-1}$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2985295, IEEE Access

being, $S_{obs} = \{x_{obs}, y_{obs}, \dot{x}_{obs}, \dot{y}_{obs}\}$. Then Equation 10, would become:

$$S = \{(x_a - x_g), (y_a - y_g), \dot{x}_a, \dot{y}_a\, x_{obs}, y_{obs}, \dot{x}_{obs}, \dot{y}_{obs}\},$$

note that we focus this work on exploring the ability of the agent to generalize its knowledge to unseen situations and multiple obstacles, to that purpose the agent is restricted to observing a single agent at each moment.

To compose the set of information necessary for acting precisely, we feed the policy a vector of observations about the environment. The vector of observation $\vec{D} = \{d_1, d_2, \ldots, d_{180}\}$ is composed of 180 observations around the robot, simulating a LIDAR sensor with 1 degree precision.

The policy π_θ, which maps the observation vector $\vec{D}$ and the state S_i to the actions A was parameterized using a neural network. The policy network is not very deep. It has only three hidden layers. Such a configuration was an implementation choice to improve the time of convergence. In order to compensate for such a shallow network, the state space $\mathcal{S}$ and observation space $\mathcal{D}$ were reduced by a design strategy, in which the policy was trained such that the agent would learn only a single behavior. The agent learns to go forward (positive y direction) while avoiding static, dynamic, and human obstacles.

Such a design choice would limit the utilization of the system in broad areas. The agent would not be able to follow goals which were not aligned in the y direction. In order to add more mobility to the agent, we map the action vector to the desired goal based on the previous waypoint and the next waypoint. The base coordinate system is rotated by ϕ, so that the direction of the goal, $\vec{G}_d = \vec{X}_k - \vec{X}_{k-1}$, coincides with the new direction of the y axis, y'.

We solve the generalization problem by performing a linear mapping in the coordinates of the system. More specifically, we calculate the angle phi as the rotation angle of $\vec{G}_d$ in polar coordinates, rotate the original coordinates system, and map $\vec{A}$ to the new coordinate system. The option for using such a rotation matrix allows the agent to cover any given direction in the 2-dimensional space, even in asymmetrical cases, conditioned to the limitations of the static graph.

IV. RESULTS

In this section, we evaluate the performance of the proposed algorithm and describe the learning process. For evaluation, in this paper, we have outlined an indoor scenario, with *a priori* known structures (external and middle walls), and unknown obstacles (boxes and people), in which the agent needs to arrive at a specific destination in a store. In this work we focus on environments that mix static and dynamic restrictions, modeled as spatio-temporal graphs. To the best of our knowledge, no environment with such conditions and modelling was available and experimented on at the time of the writing of this article. Most of the trained RL policies, degrades or do not transfer from one domain to the other, leading to a loss of performance or being completely incompatible due to the differences in the observation vector. A comparison with other learning method such as the PRM-RL in [50] would not be relevant due to the incompatibility of inputs, unavailability of the trained model, and the different nature of the proposed scenarios. As a comparison, we have implemented an artificial potential fields method (as comparison to the RL policy), as it is widely used and proven to be efficient in the literature under the conditions presented in this article and similar conditions presented in a previous article by one of the authors [31] .

A. DEEP RL AGENT TRAINING

We train our agent in a straightforward scenario composed of people(dynamic obstacles) and static obstacles using PPO [24]. We set γ to be a larger number, so that agent acts in the present in order to prepare for rewards in the distant future. We set λ to a higher value, so that the agent relies more on the current reward than the estimate it is producing. We have a Batch Size of 2048, to improve the updates in the gradient descend (more indicated for continuous systems). Our trainer setup is described in Table 1.

Parameter	Value
Batch Size	2048
β	0.005
Buffer Size	20480
Epsilon	0.2
Gamma	0.99
Hidden units	183
Lambda	0.95
Learning Rate	0.0003
Normalization	true
Time Horizon	512

TABLE 1: List of Hyper-parameters used for the Deep Reinforcement Learning agent PPO trainer.

The training environment, though virtual, simulates with high fidelity real-world physics. Under this scenario, it would be easy to feed as information to the robot the position of the obstacles. However, we have limited the knowledge of the robot to a 4m, 180 degrees LIDAR perception. This restricted implementation emulates as close as possible a real-life scenario with obstacle occlusion, sensor restrictions, and environmental restrictions. In addition to the sensor readings, the agent also knows its location in the environment (GPS emulation), and the location of the target. Both the robot's location and target location are in a global frame.

The training scenario consists of a long straight road in which the robot starts in one end, and has its target known to be on the other side of this road. The environment, constructed in Unity3D, was configured with a set of rewards $R \in \mathcal{R}$. $\mathcal{R}$ is defined in Table 2.

We have trained the algorithm for 3000000 steps. Every episode of the training would have at most 4500 steps. The cumulative reward, episode length, and policy loss are shown in Figure 4. As seen in Figure 4-(a), initially, with random policy, the agent has very erratic behavior and does not accomplish or demonstrate objective towards a goal within

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2985295, IEEE Access

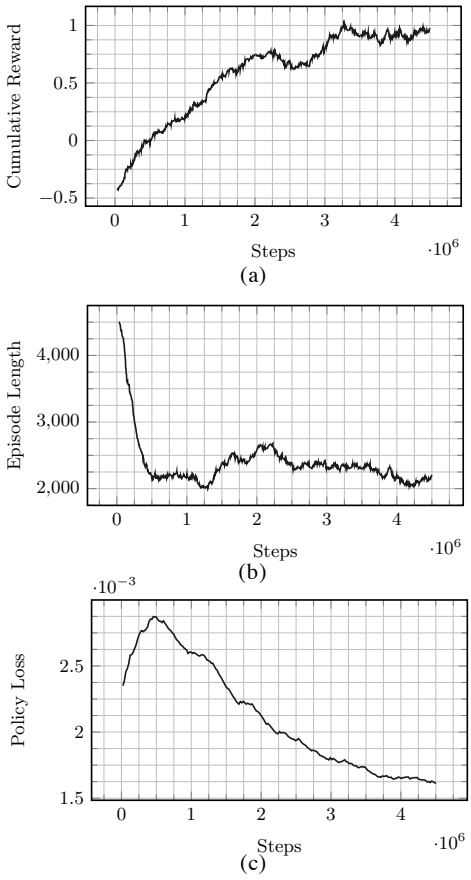


FIGURE 4: The cumulative reward in (a) grows as the episode length (b), and policy loss (c) reduces. The slight increase in around 1.25M steps can be associated with less collisions and more objective oriented behavior.

	Event	Reward
1	Achieved Goal	0.5
2	Collision to Human	-1
3	Collision to Obstacle	-0.7
4	Collision to side	-0.5
5	Velocity >0	0.001 (per frame)
6	Velocity <0	-0.0005 (per frame)

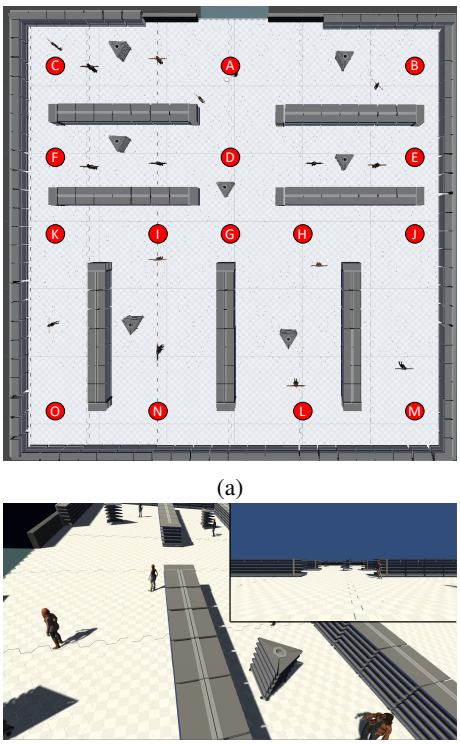
TABLE 2: Rewards defined in the environment for training. Goal achievement or collision are final actions. Rewards for positive velocity are added each frame to enforce goal seek and reduce training time.

the 4500 step limit. As the training occurs, the episode length reduces, which means the agent has more effective behavior but still collides very soon (low rewards). As it explores the environment and learns, its actions are more assertive towards the goal, which increases the received rewards. The oscillations noticed in the cumulative rewards can be interpreted as adjustments in the behavior due to the exploration/exploitation trade-off in the RL framework. In Figure 4-(c), we observe an inflection of the Policy Loss between steps 0 10^6, we associate such inflection to the observed behavior of the agent. Initially, the agent is learning about the unknown MDP and is collecting experiences by sampling actions from an almost random policy. In the initial states, the gradient of the loss function is trying to adjust the weights for actions that yields a positive reward, but as there's no effective behavior towards the goal, there's an increase in the loss of the policy.

The proposed experiment runs on a 3D simulated environment; in order to abstract the sensor observations to states and map it to an action, we use a parameterized policy. The policy is parameterized through a Neural Network and trained using PPO. In order to improve the training, and more importantly reduce the inference time, we have architected a shallow network with 3 hidden layers and 183 neurons in each layer. Such choices have allowed us to have a very small inference time. In our system, with the GPU dedicated only to render the environment, and inference being performed in an Intel i7-8700k CPU (having only 11% CPU utilization). we were able to achieve a 0.02s inference time.

B. CROWDED AREA NAVIGATION EVALUATION

We evaluate the performance of our algorithm in a simulated store environment. The store is composed of 14 different possible destinations. The only prior knowledge of the robot is the intersection map, saved in an undirected graph as



(a)

(b)

FIGURE 5: (a) A top view of the test environment with the intersections mapped to nodes in the graph G. (b) A 3D visualization of the environment with the robot's point of view in the top right corner.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2985295, IEEE Access

Destination	Avg Time(s)	Avg Distance(m)	Efficiency
B	12.46 ± 0.39	29.27 ± 1.11	0.90 ± 0.03
C	35.92 ± 8.92	37.93 ± 1.76	0.74 ± 0.05
D	24.16 ± 4.35	45.25 ± 1.82	0.97 ± 0.02
E	26.63 ± 1.16	62.00 ± 2.73	0.75 ± 0.05
F	23.88 ± 1.29	56.56 ± 3.05	0.75 ± 0.05
G	13.50 ± 0.52	31.27 ± 1.39	0.81 ± 0.04
H	34.47 ± 8.25	38.60 ± 1.67	0.91 ± 0.03
I	21.68 ± 0.86	50.58 ± 2.18	0.95 ± 0.02
J	28.50 ± 1.08	65.86 ± 2.57	0.65 ± 0.05
K	18.86 ± 1.26	45.70 ± 2.96	0.85 ± 0.04
L	54.77 ± 13.34	33.91 ± 2.49	0.81 ± 0.04
M	18.50 ± 0.53	38.63 ± 1.14	1.00 ± 0.01
N	28.97 ± 3.34	58.47 ± 2.43	0.90 ± 0.03
O	28.96 ± 1.27	67.94 ± 3.07	0.68 ± 0.05

TABLE 3: Results of APF

Destination	Avg Time(s)	Avg Distance(m)	Efficiency
B	11.21 ± 0.40	27.30 ± 1.23	0.81 ± 0.04
C	16.03 ± 0.57	37.54 ± 1.55	0.63 ± 0.05
D	18.92 ± 1.46	43.01 ± 2.36	0.82 ± 0.04
E	24.62 ± 1.85	58.85 ± 3.18	0.79 ± 0.05
F	20.65 ± 1.21	49.53 ± 2.73	0.75 ± 0.05
G	13.10 ± 0.60	31.49 ± 1.66	0.70 ± 0.05
H	16.06 ± 0.57	36.56 ± 1.36	0.80 ± 0.05
I	19.41 ± 0.85	46.24 ± 2.28	0.84 ± 0.04
J	24.17 ± 1.14	57.91 ± 2.91	0.65 ± 0.05
K	16.43 ± 1.00	41.19 ± 2.45	0.71 ± 0.05
L	14.96 ± 0.63	35.38 ± 1.76	0.69 ± 0.05
M	15.52 ± 0.55	34.02 ± 1.30	0.90 ± 0.03
N	23.15 ± 2.52	54.14 ± 2.83	0.78 ± 0.05
O	24.81 ± 1.27	59.69 ± 3.27	0.70 ± 0.05

TABLE 4: Results of RL

described in subsection III-A. In between waypoints the static obstacles are placed randomly, and the human behavior is also random but follows maximum speed limitation, we have simulated the crowd with random speeds ranging from 0.7km/h to 3.7km/h. The current scenario, proposed to the tests, was discretized in 2376 nodes. From these nodes, 336 nodes were occupied by the shelves (static obstacles). The other 2040 nodes were available paths to the agent and the dynamic obstacles which compose the unknown dynamic graph G_d. In order to navigate along this environment, a 15 nodes graph was designed as the static graph G_s, and assumed to be known by the agent. Under such circumstances, not considering the position the nodes occupied by the shelves, we have a ratio of 15/2040 when comparing the static graph to the dynamic.

In order to have a comprehensive evaluation of our algorithm, we have executed all possible paths (set each node as a destination once) 300 times each for a total of 4200 trials. We have compared our system with artificial potential fields (APF) as proposed in [31] with the addition of a rotation field, as shown in [66]. In the implemented controller $K_{attraction} = 0.5$,$K_{repulsion} = 25$, obstacle radius of influence and attraction switch distance are 4m. The gains were optimized for the current configuration of the store.

Table 4 and Table 3 contains the results obtained after 300 trials for each possible destination, following the waypoints generated by the A* algorithm. The average times, distances, and efficiencies for each path were calculated for the N samples with a confidence level of 95%, as follows:

$$\text{Avg Time} = \frac{1}{N}\sum_{i=1}^{N} time_i \pm 1.96 * \frac{\sigma}{\sqrt{N}}$$

$$\text{Avg Distance} = \frac{1}{N}\sum_{i=1}^{N} distance_i \pm 1.96 * \frac{\sigma}{\sqrt{N}}$$

$$\text{Efficiency} = \hat{p} \pm 1.96 * \sqrt{\frac{\hat{p} * (1-\hat{p})}{N}}$$

where σ is the standard deviation of the samples, and the efficiency rate, $\hat{p}$ is calculated by dividing the number of goal achievements by the total number of samples, $N = 300$.

In most of the cases, the algorithm running with APF had a higher achievement rate compared to its RL counterpart, but it does not translate into overall better performance. In the APF case, the gains for the algorithm were carefully optimized to the situation of the tests, in which the speed of dynamic obstacles and number of static obstacles were controlled.

This posted it remains to evaluate the efficiency of the algorithm in adversarial situations. Mainly when those vary from the training conditions of the algorithm.

C. ADVERSE CONDITIONS EVALUATION

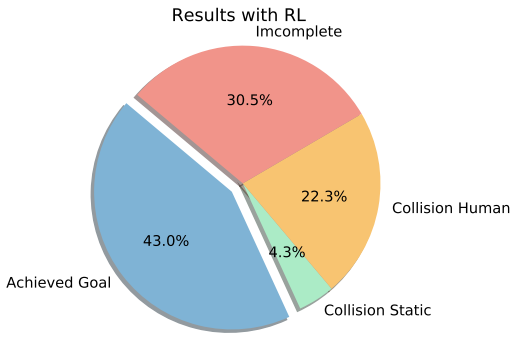
The advantage of using a non-deterministic policy over the actions is that the agent will have the possibility of better estimating its state and generalizing its distribution of actions over states, to choose the adequate action in the presented scenario. We have empirically evaluated this statement through a series of experiments.

To explore the generalization ability of the policy, we have restricted the scenario to the navigation between two waypoints and varied the maximum speed of the agent, the maximum speed of humans, the number of static obstacles and also the number of humans in the scenario. Each of the conditions was tested five times, with random placement of humans and obstacles.

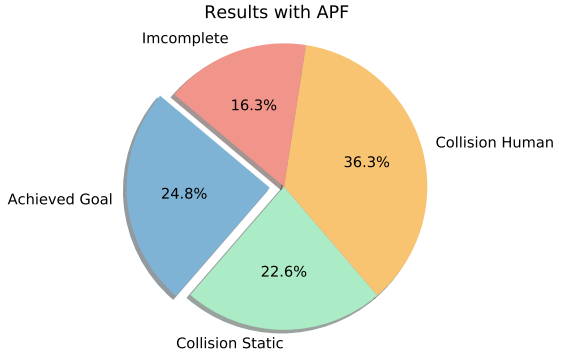
As seen in Figure 6, overall, the Reinforcement Learning algorithm had a better efficiency in the more adverse scenarios. In this case, the trials labeled as incomplete falls under the category in which the time limit for the execution was reached, without the agent achieving the goal or colliding which is the result of local minima or early termination of the process.

The maximum speed of the agent defines the time it has to react to the perceived obstacles in the environment. As the velocity increases, the reaction time reduces, and the control policy needs to take actions accounting for this fact. In the multiple path scenario, we have executed the tests with a pre-defined maximum speed, and have optimized the APF algorithm for that speed. Such circumstances, have shown APF as a more goal efficient algorithm. In Figure 7, we have isolated the different maximum velocities we have used to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2985295, IEEE Access



(a)



(b)

FIGURE 6: Overall result of the RL policy, in 9680 trials with 1936 different scenarios.(a) The Reinforcement Learning Results. (b) The APF results.

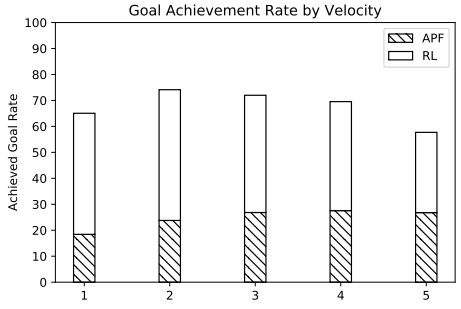


FIGURE 7: The goal achievement rate obtained, in different maximum velocities for the agent. The APF policy efficiency (in blue) decreases exponentially with the speed, opposed to the RL policy (in orange) which adapts better to the parameter variation.

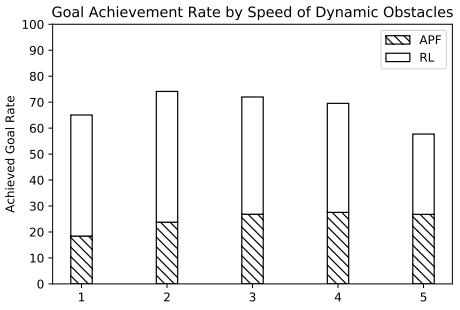


FIGURE 8: The goal achievement rate obtained, in different maximum speed of the dynamic obstacles for the agent. The RL perform better in all tests, we observe greater correlation of the dynamic obstacles speed and the efficiency of the agent.

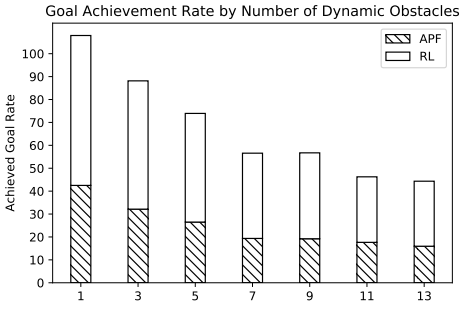


FIGURE 9: The goal achievement rate obtained, with different number of dynamic obstacles, by the agent.

test the robot and varied all the other parameters. We can see that as speed increases, the success rate of the APF method is reduced exponentially to below 20%. On the contrary, the probabilistic policy adapts better to the increase of speed.

Differently, in Figure 8, it is noticeable that even with a more significant achievement rate of the RL policy, the RL policy is greatly affected by the speed of the dynamic obstacles. We attribute this to the fact that it requires a faster lateral reaction from the agent, which was not exploited in the training phase. The lateral reaction is compromised due to the use of only a 180 degree LIDAR; it reduces in 50% the number of visualizations of the side obstacles. In this case, since the dynamic obstacles and humans are not simulated with object avoidance, they would still go in the direction of the robot.

As we see in Figures 9 and 10, as we increase the number of obstacles, either dynamical or static, the possibilities to avoid them decrease; besides, the performance of both algorithms decline as the number of obstacles in the path increase. It is evident that even with this variation and increase in complexity, the RL can have better efficiency. Overall we notice that the implemented policy can generalize the environment better and take more effective actions, to avoid obstacles, and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2985295, IEEE Access

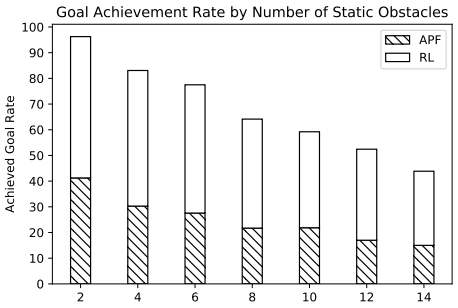


FIGURE 10: The goal achievement rate obtained, with different number of static obstacles, by the agent.

consequently achieve the goal.

V. CONCLUSION

Graph navigation for short path queries is essential for autonomous vehicles and robotic navigation. This paper studied a hybrid methodology to improve navigation efficiency in uncertain environments. The algorithm is divided into two main parts, a graph navigator based on A^* which enables navigation between landmarks and known structures of the environment, and a local navigator that abstracts the dynamics of the agent and the environment into states and probabilistically chooses the adequate action for the current state. We have extensively tested our algorithm in many adverse scenarios. Initially, we tested for different paths in a simulated store scenario and compared its performance to the Artificial Potential Fields method, optimized for those specific conditions. Also, we tested and compared both methods when applied to adverse situations, to which they were neither trained nor optimized. The algorithm based on the Reinforcement Learning Policy, outperformed the APF one in up to 100% in specific cases.

Our future work will focus on the integration of the performance analysis of the algorithm on the time of training, in order to improve the training phase of the algorithm.

REFERENCES

[1] G. Alanis-Lobato, M. A. Andrade-Navarro, and M. H. Schaefer, "Hippie v2. 0: enhancing meaningfulness and reliability of protein–protein interaction networks," Nucleic acids research, p. gkw985, 2016.

[2] B. Lwowski, P. Rad, and K.-K. R. Choo, "Geospatial event detection by grouping emotion contagion in social media," IEEE Transactions on Big Data, 2018.

[3] H. Chen, X. Chen, and H. Liu, "How does language change as a lexical network? an investigation based on written chinese word co-occurrence networks," PloS one, vol. 13, no. 2, p. e0192545, 2018.

[4] A. Mandalika, O. Salzman, and S. Srinivasa, "Lazy receding horizon a* for efficient path planning in graphs with expensive-to-evaluate edges," in Twenty-Eighth International Conference on Automated Planning and Scheduling, 2018.

[5] S. Ma, J. Li, C. Hu, X. Lin, and J. Huai, "Big graph search: challenges and techniques," Frontiers of Computer Science, vol. 10, no. 3, pp. 387–398, 2016.

[6] M. Bloznelis and L. Leskelä, "Diclique clustering in a directed random graph," in International Workshop on Algorithms and Models for the Web-Graph. Springer, 2016, pp. 22–33.

[7] G. Ramalingam and T. Reps, "On the computational complexity of dynamic graph problems," Theoretical Computer Science, vol. 158, no. 1-2, pp. 233–277, 1996.

[8] O. Reingold, "Undirected connectivity in log-space," Journal of the ACM (JACM), vol. 55, no. 4, p. 17, 2008.

[9] M. Likhachev, D. I. Ferguson, G. J. Gordon, A. Stentz, and S. Thrun, "Anytime dynamic a*: An anytime, replanning algorithm." in ICAPS, vol. 5, 2005, pp. 262–271.

[10] S. S. Ge and Y. J. Cui, "New potential functions for mobile robot path planning," IEEE Transactions on robotics and automation, vol. 16, no. 5, pp. 615–620, 2000.

[11] D. F. W. B. S. Thrun, D. Fox, and W. Burgard, "The dynamic window approach to collision avoidance," IEEE Transactions on Robotics and Automation, vol. 4, p. 1, 1997.

[12] P. Raja and S. Pugazhenthi, "Optimal path planning of mobile robots: A review," International Journal of Physical Sciences, vol. 7, no. 9, pp. 1314–1320, 2012.

[13] Y. F. Chen, M. Everett, M. Liu, and J. P. How, "Socially aware motion planning with deep reinforcement learning," in 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2017, pp. 1343–1350.

[14] S. Arumugam, I. Hamid, and V. Abraham, "Decomposition of graphs into paths and cycles," Journal of Discrete Mathematics, vol. 2013, 2013.

[15] T. Opsahl, F. Agneessens, and J. Skvoretz, "Node centrality in weighted networks: Generalizing degree and shortest paths," Social networks, vol. 32, no. 3, pp. 245–251, 2010.

[16] T. Nie, Z. Guo, K. Zhao, and Z.-M. Lu, "Using mapping entropy to identify node centrality in complex networks," Physica A: Statistical Mechanics and its Applications, vol. 453, pp. 290–297, 2016.

[17] A. Stentz, "Optimal and efficient path planning for partially known environments," in Intelligent Unmanned Ground Vehicles. Springer, 1997, pp. 203–220.

[18] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," IEEE transactions on Systems Science and Cybernetics, vol. 4, no. 2, pp. 100–107, 1968.

[19] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[20] A. Yershova, L. Jaillet, T. Siméon, and S. M. LaValle, "Dynamic-domain rrts: Efficient exploration by controlling the sampling domain," in 2005 IEEE International Conference on Robotics and Automation. IEEE, 2005, pp. 3856–3861.

[21] L. Kavraki, P. Svestka, and M. H. Overmars, Probabilistic roadmaps for path planning in high-dimensional configuration spaces. Unknown Publisher, 1994, vol. 1994.

[22] K. Keahey, P. Riteau, D. Stanzione, T. Cockerill, J. Manbretti, P. Rad, and R. Paul, "Chameleon: a scalable production testbed for computer science research," Contemporary High Performance Computing, vol. 3, 2017.

[23] C. A. Stewart, D. Y. Hancock, T. Miller, J. Fischer, R. L. Liming, G. Turner, J. M. Lowe, S. Gregory, E. Skidmore, M. Vaughn et al., "Jetstream: A novel cloud system for science," in Contemporary High Performance Computing. CRC Press, 2019, pp. 189–222.

[24] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," arXiv preprint arXiv:1707.06347, 2017.

[25] Q. Yang and S.-J. Yoo, "Optimal uav path planning: Sensing data acquisition over iot sensor networks using multi-objective bio-inspired algorithms," IEEE Access, vol. 6, pp. 13 671–13 684, 2018.

[26] C. M. Eaton, E. K. Chong, and A. A. Maciejewski, "Robust uav path planning using pomdp with limited fov sensor," in Control Technology and Applications (CCTA), 2017 IEEE Conference on. IEEE, 2017, pp. 1530–1535.

[27] S. H. Silva, P. Rad, N. Beebe, K.-K. R. Choo, and M. Umapathy, "Cooperative unmanned aerial vehicles with privacy preserving deep vision for real-time object identification and tracking," Journal of Parallel and Distributed Computing, vol. 131, pp. 147–160, 2019.

[28] Y. Singh, S. Sharma, R. Sutton, D. Hatton, and A. Khan, "A constrained a* approach towards optimal path planning for an unmanned surface vehicle in a maritime environment containing dynamic obstacles and ocean currents," Ocean Engineering, vol. 169, pp. 187–201, 2018.

[29] J. Ni, L. Wu, P. Shi, and S. X. Yang, "A dynamic bioinspired neural network based real-time path planning method for autonomous underwater vehicles," Computational intelligence and neuroscience, vol. 2017, 2017.

[30] Y. Li, T. Ma, P. Chen, Y. Jiang, R. Wang, and Q. Zhang, "Autonomous underwater vehicle optimal path planning method for seabed terrain matching navigation," Ocean Engineering, vol. 133, pp. 107–115, 2017.
[31] R. R. Da Silva, S. Silva, G. Dubrovskiy, and H. Lin, "Safeguardpf: Safety guaranteed reactive potential fields for mobile robots in unknown and dynamic environments," arXiv preprint arXiv:1609.07006, 2016.
[32] M. Li, C. Wang, Z. Chen, X. Lu, M. Wu, and P. Hou, "Path planning of mobile robot based on genetic algorithm and gene rearrangement," in Chinese Automation Congress (CAC), 2017. IEEE, 2017, pp. 6999–7004.
[33] B. Fu, L. Chen, Y. Zhou, D. Zheng, Z. Wei, J. Dai, and H. Pan, "An improved a* algorithm for the industrial robot path planning with high success rate and short length," Robotics and Autonomous Systems, vol. 106, pp. 26–37, 2018.
[34] M. W. Otte, "A survey of machine learning approaches to robotic path-planning," Cs. Colorado. Edu, 2015.
[35] S. M. LaValle, Planning algorithms. Cambridge university press, 2006.
[36] S.-H. Hsu, S.-H. Chan, P.-T. Wu, K. Xiao, and L.-C. Fu, "Distributed deep reinforcement learning based indoor visual navigation," in 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2018, pp. 2532–2537.
[37] A. K. Guruji, H. Agarwal, and D. Parsediya, "Time-efficient a* algorithm for robot path planning," Procedia Technology, vol. 23, pp. 144–149, 2016.
[38] S. Alnasser and H. Bennaceur, "An efficient genetic algorithm for the global robot path planning problem," in Digital Information and Communication Technology and its Applications (DICTAP), 2016 sixth international conference on. IEEE, 2016, pp. 97–102.
[39] J. Zhang, Y. Xia, and G. Shen, "A novel learning-based global path planning algorithm for planetary rovers," arXiv preprint arXiv:1811.10437, 2018.
[40] M. Shahvali, M.-B. Naghibi-Sistani, and J. Askari, "Adaptive output-feedback bipartite consensus for nonstrict-feedback nonlinear multi-agent systems: A finite-time approach," Neurocomputing, vol. 318, pp. 7–17, 2018.
[41] M. Shahvali, N. Pariz, and M. Akbariyan, "Distributed finite-time control for arbitrary switched nonlinear multi-agent systems: an observer-based approach," Nonlinear Dynamics, vol. 94, no. 3, pp. 2127–2142, 2018.
[42] M. Shahvali, M.-B. Naghibi-Sistani, and H. Modares, "Distributed consensus control for a network of incommensurate fractional-order systems," IEEE Control Systems Letters, vol. 3, no. 2, pp. 481–486, 2019.
[43] P. K. Mohanty and D. R. Parhi, "Optimal path planning for a mobile robot using cuckoo search algorithm," Journal of Experimental & Theoretical Artificial Intelligence, vol. 28, no. 1-2, pp. 35–52, 2016.
[44] T. Hwu, A. Y. Wang, N. Oros, and J. L. Krichmar, "Adaptive robot path planning using a spiking neuron algorithm with axonal delays," IEEE Transactions on Cognitive and Developmental Systems, vol. 10, no. 2, pp. 126–137, 2018.
[45] A. Pandey and D. R. Parhi, "Optimum path planning of mobile robot in unknown static and dynamic environments using fuzzy-wind driven optimization algorithm," Defence Technology, vol. 13, no. 1, pp. 47–58, 2017.
[46] M. A. Hossain and I. Ferdous, "Autonomous robot path planning in dynamic environment using a new optimization technique inspired by bacterial foraging technique," Robotics and Autonomous Systems, vol. 64, pp. 137–141, 2015.
[47] Y.-b. Chen, G.-c. Luo, Y.-s. Mei, J.-q. Yu, and X.-l. Su, "Uav path planning using artificial potential field method updated by optimal control theory," International Journal of Systems Science, vol. 47, no. 6, pp. 1407–1420, 2016.
[48] N. Hirose, A. Sadeghian, P. Goebel, and S. Savarese, "To go or not to go? a near unsupervised learning approach for robot navigation," arXiv preprint arXiv:1709.05439, 2017.
[49] G. Wei, D. Hus, W. S. Lee, S. Shen, and K. Subramanian, "Intention-net: Integrating planning and deep learning for goal-directed autonomous navigation," arXiv preprint arXiv:1710.05627, 2017.
[50] A. Faust, K. Oslund, O. Ramirez, A. Francis, L. Tapia, M. Fiser, and J. Davidson, "Prm-rl: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning," in 2018 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2018, pp. 5113–5120.
[51] T. Fan, X. Cheng, J. Pan, D. Monacha, and R. Yang, "Crowdmove: Autonomous mapless navigation in crowded scenarios," arXiv preprint arXiv:1807.07870, 2018.
[52] M. Fahad, Z. Chen, and Y. Guo, "Learning how pedestrians navigate: A deep inverse reinforcement learning approach," in 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2018, pp. 819–826.
[53] M. Everett, Y. F. Chen, and J. P. How, "Motion planning among dynamic, decision-making agents with deep reinforcement learning," in 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2018, pp. 3052–3059.
[54] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction. MIT press, 2018.
[55] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," arXiv preprint arXiv:1509.02971, 2015.
[56] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in International Conference on Machine Learning, 2015, pp. 1889–1897.
[57] S. Kakade and J. Langford, "Approximately optimal approximate reinforcement learning," in ICML, vol. 2, 2002, pp. 267–274.
[58] T. Van Erven and P. Harremos, "Rényi divergence and kullback-leibler divergence," IEEE Transactions on Information Theory, vol. 60, no. 7, pp. 3797–3820, 2014.
[59] T. Uras and S. Koenig, "Fast near-optimal path planning on state lattices with subgoal graphs," in Eleventh Annual Symposium on Combinatorial Search, 2018.
[60] H. Van Seijen, M. Fatemi, J. Romoff, R. Laroche, T. Barnes, and J. Tsang, "Hybrid reward architecture for reinforcement learning," in Advances in Neural Information Processing Systems, 2017, pp. 5392–5402.
[61] P. Marin-Plaza, A. Hussein, D. Martin, and A. d. l. Escalera, "Global and local path planning study in a ros-based research platform for autonomous vehicles," Journal of Advanced Transportation, vol. 2018, 2018.
[62] G. Hollinger, S. Singh, J. Djugash, and A. Kehagias, "Efficient multi-robot search for a moving target," The International Journal of Robotics Research, vol. 28, no. 2, pp. 201–219, 2009.
[63] S. M. Dibaji and H. Ishii, "Consensus of second-order multi-agent systems in the presence of locally bounded faults," Systems & Control Letters, vol. 79, pp. 23–29, 2015.
[64] C. Chen, Y. Liu, S. Kreiss, and A. Alahi, "Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning," arXiv preprint arXiv:1809.08835, 2018.
[65] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in Advances in neural information processing systems, 2000, pp. 1057–1063.
[66] J. Sfeir, M. Saad, and H. Saliah-Hassane, "An improved artificial potential field approach to real-time mobile robot path planning in an unknown environment," in 2011 IEEE international symposium on robotic and sensors environments (ROSE). IEEE, 2011, pp. 208–213.

SAMUEL HENRIQUE SILVA is currently a Ph.D. student and research fellow at the Open Cloud Institute of University of Texas at San Antonio (UTSA), San Antonio, TX, USA. Samuel received the Bachelor of Science (B.Sc.) degree in Control and Automation Engineering from State University of Campinas, Campinas, Brazil, in 2012 and the M.S. degree in Electrical Engineering from the University of Notre Dame, Notre Dame, IN, USA in 2016. He is a member of the IEEE, Eta Kappa Nu honor society. Samuel's research interests are in the areas of artificial intelligence, control systems, autonomous decision making, multi-agent systems and adversarial environments.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2985295, IEEE Access

ADEL ALAEDDINI is an Associate Professor of Mechanical Engineering at University of Texas at San Antonio. He obtained his PhD in Industrial and Systems Engineering from Wayne State University. He also did a post doc at University of Michigan, Ann Arbor. His main research interests include statistical learning in systems modeling and control, and data analytics in health care and manufacturing. He has contributed to over 30 peer-reviewed publications in journals such as IISE Transactions, Production and Operations Management (POMS), and Information Sciences.

PAUL RAD is a co-founder and Associate Director of the Open Cloud Institute (OCI), and an Associate Professor with the Information Systems and Cyber Security Department at the University of Texas at San Antonio. He received his first B.S. degree from Sharif University of Technology in Computer Engineering in 1994, his 1st master in artificial intelligence from the Tehran Polytechnic, his 2nd master in computer science from the University of Texas at San Antonio (Magna Cum Laude) in 1999, and his Ph.D. in electrical and computer engineering from the University of Texas at San Antonio. He was a recipient of the Most Outstanding Graduate Student in the College of Engineering, 2016, earned the Rackspace Innovation Mentor Program Award for establishing Rackspace patent community board structure and mentoring employees (2012), earned the Dell Corporation Company Excellence (ACE) Award for exceptional performance and innovative product research and development contributions (2007), and earned the Dell Inventor Milestone Award, Top 3 Dell Inventor of the year (2005). He holds 15 U.S. patents on cyber infrastructure, cloud computing, and big data analytics with over 300 product citations by top fortune 500 leading technology companies such as Amazon, Microsoft, IBM, Cisco, Amazon Technologies, HP, and VMware. He has advised over 200 companies on cloud computing and data analytics with over 50 keynote presentations. He serves on the advisory board for several startups, high performance cloud group chair at the Cloud Advisory Council (CAC), OpenStack Foundation Member (the #1 open source cloud software), San Antonio Tech Bloc Founding Member, and ChildrenâĂŹs Hospital of San Antonio Foundation board member.